CertiCrypt

Language-Based Cryptographic Proofs in Coq

Gilles Barthe[1,2] Benjamin Grégoire[1,3]
Santiago Zanella [1,3]

[1]Microsoft Research - INRIA Joint Centre, France

CENTRE DE RECHERCHE COMMUN INRIA MICROSOFT RESEARCH

[2]IMDEA Software, Madrid, Spain

imdea instituto madrileño de estudios avanzados

[3]INRIA Sophia Antipolis - Méditerranée, France

INRIA SOPHIA ANTIPOLIS

POPL 2009

What's wrong with cryptographic proofs?

- *In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor*
  M. Bellare and P. Rogaway.
- *Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect)*
  S. Halevi
- *Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify*
  V. Shoup

What's wrong with cryptographic proofs?

- *In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor*
  M. Bellare and P. Rogaway.
- *Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect)*
  S. Halevi
- *Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify*
  V. Shoup

Game-based cryptographic proofs

Attack Game

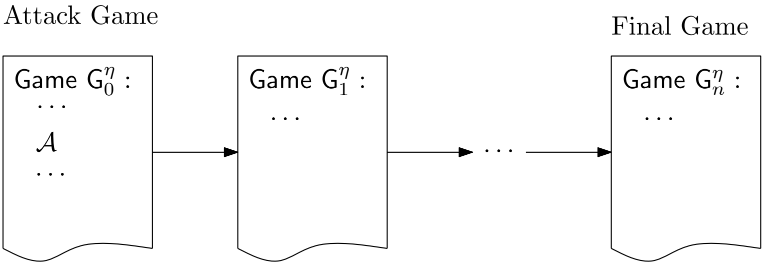
Game G_0^η :
$\ldots$
$\mathcal{A}$
$\ldots$

$\Pr_{\mathsf{G}_0^\eta}[A_0]$

$$\Pr_{\mathsf{G}_0^\eta}[A_0] \leq \epsilon(\eta)$$

Security property

Game-based cryptographic proofs

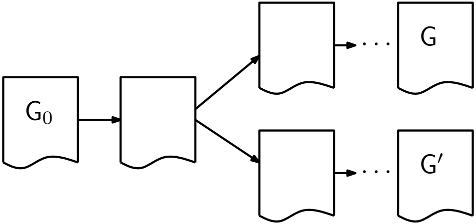


$$\Pr_{G_0^\eta}[A_0] \quad \leq \quad h_1(\Pr_{G_1^\eta}[A_1]) \quad \leq \quad \cdots \quad \leq \quad h_n(\Pr_{G_n^\eta}[A_n])$$

$$\Pr_{G_0^\eta}[A_0] \leq h(\Pr_{G_n^\eta}[A_n]) \leq \epsilon(\eta)$$

Game-based proofs: essence and problems

Independent events



$$\Pr_{G_0}[A_0] \leq h(\Pr_{G}[A]) \times h'(\Pr_{G'}[A'])$$

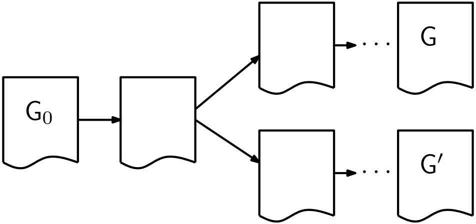
Essence: relate the probability of events in consecutive games

But,

- How do we represent games?
- What adversaries are *feasible*?
- How do we make a proof hold for any feasible adversary?

Game-based proofs: essence and problems

Independent events



$$\Pr_{\mathsf{G}_0}[A_0] \leq h(\Pr_{\mathsf{G}}[A]) \times h'(\Pr_{\mathsf{G}'}[A'])$$

Essence: relate the probability of events in consecutive games

But,

- How do we represent games?
- What adversaries are *feasible*?
- How do we make a proof hold for any feasible adversary?

Language-based proofs

What if we represent games as programs?

Games	$\Longrightarrow$	programs
Probability space	$\Longrightarrow$	program denotation
Game transformations	$\Longrightarrow$	program transformations
Generic adversary	$\Longrightarrow$	unspecified procedure
Feasibility	$\Longrightarrow$	Probabilistic Polynomial-Time

pWHILE: a probabilistic programming language

$$
\begin{array}{llll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
 & \mid & \mathcal{V} \xleftarrow{\$} \mathcal{D} & \text{random sampling} \\
 & \mid & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
 & \mid & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
 & \mid & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \text{nil} & \text{nop} \\
 & \mid & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}
$$

Measure monad: $M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$$

$[\![x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}]\!]\ m\ =$

Probability: $\Pr_{G,m}[A] \stackrel{\text{def}}{=} [\![G]\!]\ m\ \mathbb{1}_A$

pWHILE: a probabilistic programming language

$\mathcal{I}$	$::=$	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	$\mid$	$\mathcal{V} \xleftarrow{\$} \mathcal{D}$	random sampling
	$\mid$	if $\mathcal{E}$ then $\mathcal{C}$ else $\mathcal{C}$	conditional
	$\mid$	while $\mathcal{E}$ do $\mathcal{C}$	while loop
	$\mid$	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call
$\mathcal{C}$	$::=$	nil	nop
	$\mid$	$\mathcal{I};\ \mathcal{C}$	sequence

Measure monad: $M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$$

$$\begin{aligned} &[\![x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}]\!]\ m\ f = \\ &\quad \tfrac{1}{4}\ f(m[0,0/x,y]) \quad + \quad \tfrac{1}{4}\ f(m[0,1/x,y]) \quad + \\ &\quad \tfrac{1}{4}\ f(m[1,0/x,y]) \quad + \quad \tfrac{1}{4}\ f(m[1,1/x,y]) \end{aligned}$$

Probability: $\Pr_{G,m}[A] \stackrel{\text{def}}{=} [\![G]\!]\ m\ \mathbb{1}_A$

pWHILE: a probabilistic programming language

$\mathcal{I}$	$::=$	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	$\mid$	$\mathcal{V} \xleftarrow{\$} \mathcal{D}$	random sampling
	$\mid$	if $\mathcal{E}$ then $\mathcal{C}$ else $\mathcal{C}$	conditional
	$\mid$	while $\mathcal{E}$ do $\mathcal{C}$	while loop
	$\mid$	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call
$\mathcal{C}$	$::=$	nil	nop
	$\mid$	$\mathcal{I};\ \mathcal{C}$	sequence

Measure monad: $M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$$

$$\begin{aligned}[\![x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}]\!]\ m\ \mathbb{1}_{x \neq y} = \\ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[0,0/x,y]) \ + \ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[0,1/x,y]) \ + \\ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[1,0/x,y]) \ + \ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[1,1/x,y])\end{aligned}$$

Probability: $\Pr_{G,m}[A] \stackrel{\text{def}}{=} [\![G]\!]\ m\ \mathbb{1}_A$

pWHILE: a probabilistic programming language

$$
\begin{array}{lcll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
& | & \mathcal{V} \xleftarrow{\$} \mathcal{D} & \text{random sampling} \\
& | & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
& | & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \text{nil} & \text{nop} \\
& | & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}
$$

Measure monad: $M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$$

$$
\begin{array}{l}
[\![x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}]\!]\ m\ \mathbb{1}_{x \neq y} = \\
\qquad 0 \qquad + \quad \frac{1}{4} \quad + \\
\qquad \frac{1}{4} \qquad + \quad 0
\end{array}
$$

Probability: $\Pr_{G,m}[A] \stackrel{\text{def}}{=} [\![G]\!]\ m\ \mathbb{1}_A$

pWHILE: a probabilistic programming language

$$
\begin{array}{rcll}
\mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
& | & \mathcal{V} \stackrel{\$}{\leftarrow} \mathcal{D} & \text{random sampling} \\
& | & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
& | & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} & ::= & \text{nil} & \text{nop} \\
& | & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}
$$

Measure monad: $M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$

$$[\![\cdot]\!] : \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$$

$$[\![x \stackrel{\$}{\leftarrow} \{0,1\};\ y \stackrel{\$}{\leftarrow} \{0,1\}]\!]\ m\ \mathbb{1}_{x \neq y} = \frac{1}{2}$$

Probability: $\Pr_{\mathrm{G},m}[A] \stackrel{\text{def}}{=} [\![\mathrm{G}]\!]\ m\ \mathbb{1}_A$

Untyped vs. typed language

- 1st attempt: untyped language, lots of problems
  - No guarantee that programs are well-typed
  - Had to deal with ill-typed programs
- 2nd attempt: typed language (dependently typed syntax!)
  - Programs are well-typed by construction

$$
\begin{array}{l}
\textbf{Inductive}\ \mathcal{I} : \textbf{Type} := \\
\mid \text{Assign} : \forall t,\ \mathcal{V}_t \to \mathcal{E}_t \to \mathcal{I} \\
\mid \text{Rand} \quad : \forall t,\ \mathcal{V}_t \to \mathcal{D}_t \to \mathcal{I} \\
\mid \text{Cond} \quad : \mathcal{E}_{\text{Bool}} \to \mathcal{C} \to \mathcal{C} \to \mathcal{I} \\
\mid \text{While} \quad : \mathcal{E}_{\text{Bool}} \to \mathcal{C} \to \mathcal{I} \\
\mid \text{Call} \quad : \forall l\ t, \mathcal{P}_{(l,t)} \to \mathcal{V}_t \to \mathcal{E}_l^{\star} \to \mathcal{I} \\
\textbf{where}\ \mathcal{C} := \mathcal{I}^{\star}.
\end{array}
$$

Parametrized semantics: $[\![\cdot]\!] : \forall \eta,\ \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$

Untyped vs. typed language

- 1^{st} attempt: untyped language, lots of problems
  - No guarantee that programs are well-typed
  - Had to deal with ill-typed programs
- 2^{nd} attempt: typed language (dependently typed syntax!)
  - Programs are well-typed by construction

$$
\begin{array}{l}
\textbf{Inductive } \mathcal{I} : \textbf{Type} := \\
\mid \text{Assign} : \forall t,\ \mathcal{V}_t \to \mathcal{E}_t \to \mathcal{I} \\
\mid \text{Rand} \quad : \forall t,\ \mathcal{V}_t \to \mathcal{D}_t \to \mathcal{I} \\
\mid \text{Cond} \quad : \mathcal{E}_{\text{Bool}} \to \mathcal{C} \to \mathcal{C} \to \mathcal{I} \\
\mid \text{While} \quad : \mathcal{E}_{\text{Bool}} \to \mathcal{C} \to \mathcal{I} \\
\mid \text{Call} \quad : \forall l\ t, \mathcal{P}_{(l,t)} \to\ \mathcal{V}_t \to \mathcal{E}_l^\star \to \mathcal{I} \\
\textbf{where } \mathcal{C} := \mathcal{I}^\star .
\end{array}
$$

Parametrized semantics: $[\![\cdot]\!] : \forall \eta,\ \mathcal{C} \to \mathcal{M} \to M(\mathcal{M})$

Characterizing feasible adversaries

A cost model for reasoning about program complexity

$$[\![\cdot]\!]' : \forall \eta,\ \mathcal{C} \to (\mathcal{M} \times \mathbb{N}) \to M(\mathcal{M} \times \mathbb{N})$$

Non-intrusive:

$$[\![\mathrm{G}]\!]\ m = \mathrm{bind}\ ([\![\mathrm{G}]\!]'\ (m, 0))\ (\lambda mn.\ \mathrm{unit}\ (\mathrm{fst}\ mn))$$

A program G runs in probabilistic polynomial time if:

- It terminates with probablity 1 (i.e. $\forall m,\ \Pr_{\mathrm{G},m}[\mathrm{true}] = 1$)
- There exists a polynomial $p(\cdot)$ s.t. if (m', n) is reachable with positive probability, then $n \leq p(\eta)$

Characterizing feasible adversaries

A cost model for reasoning about program complexity

$$[\![\cdot]\!]' : \forall \eta,\ \mathcal{C} \to (\mathcal{M} \times \mathbb{N}) \to M(\mathcal{M} \times \mathbb{N})$$

Non-intrusive:

$$[\![\mathrm{G}]\!]\ m = \mathrm{bind}\ ([\![\mathrm{G}]\!]'\ (m, 0))\ (\lambda mn.\ \mathrm{unit}\ (\mathrm{fst}\ mn))$$

A program G runs in probabilistic polynomial time if:

- It terminates with probablity 1 (i.e. $\forall m,\ \Pr_{\mathrm{G},m}[\mathrm{true}] = 1$)
- There exists a polynomial $p(\cdot)$ s.t. if (m', n) is reachable with positive probability, then $n \leq p(\eta)$

Program equivalence

Definition (Observational equivalence)

$$f =_X g \stackrel{\text{def}}{=} \forall m_1\ m_2,\ m_1(X) = m_2(X) \implies f\ m_1 = g\ m_2$$

$$\models \mathrm{G}_1 \simeq^I_O \mathrm{G}_2 \stackrel{\text{def}}{=} \forall m_1\ m_2\ f\ g,\ m_1(I) = m_2(I) \land f =_O g \implies [\![\mathrm{G}_1]\!]\ m_1\ f = [\![\mathrm{G}_2]\!]\ m_2\ g$$

Generalizes information flow security.
But is not general enough...

$$\frac{???}{\models \text{if } x = 0 \text{ then } y \leftarrow x \text{ else } y \leftarrow 1 \simeq^{\{x\}}_{\{x,y\}} \text{if } x = 0 \text{ then } y \leftarrow 0 \text{ else } y \leftarrow 1}$$

IMDEA — Centre de Recherche Commun INRIA Microsoft Research — INRIA Sophia Antipolis

Program equivalence

Definition (Observational equivalence)

$$f =_X g \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2,\ m_1(X) = m_2(X) \implies f\ m_1 = g\ m_2$$

$$\models \mathsf{G}_1 \simeq^I_O \mathsf{G}_2 \quad \stackrel{\text{def}}{=} \quad \forall m_1\ m_2\ f\ g,\ m_1(I) = m_2(I)\ \wedge\ f =_O g \implies [\![\mathsf{G}_1]\!]\ m_1\ f = [\![\mathsf{G}_2]\!]\ m_2\ g$$

Generalizes information flow security.
But is not general enough...

$$\frac{???}{\models \text{if } x = 0 \text{ then } y \leftarrow x \text{ else } y \leftarrow 1 \simeq^{\{x\}}_{\{x,y\}} \text{if } x = 0 \text{ then } y \leftarrow 0 \text{ else } y \leftarrow 1}$$

Program equivalence

Definition (Observational equivalence, generalization)

$\vDash G_1 \sim G_2 : \Psi \Rightarrow \Phi \stackrel{\text{def}}{=}$

$$\forall\, m_1\ m_2.\ m_1\ \Psi\ m_2 \Rightarrow [\![G_1]\!]\ m_1 \sim_\Phi [\![G_2]\!]\ m_2$$

Where $\sim_\Phi$ is the lifting of relation Φ from memories to distributions.

$$\frac{\begin{array}{l}(x = 0) \sim_{\{x\}} (x = 0) \\ \vDash y \leftarrow x \sim y \leftarrow 0 : =_{\{x\}} \wedge (x = 0)\langle 1\rangle \Rightarrow =_{\{x,y\}} \\ \vDash y \leftarrow 1 \sim y \leftarrow 1 : =_{\{x\}} \wedge (x \neq 0)\langle 1\rangle \Rightarrow =_{\{x,y\}}\end{array}}{\begin{array}{l}\text{if } x = 0 \text{ then } y \leftarrow x \text{ else } y \leftarrow 1 \sim \\ \text{if } x = 0 \text{ then } y \leftarrow 0 \text{ else } y \leftarrow 1 : =_{\{x\}} \Rightarrow =_{\{x,y\}}\end{array}}$$

From program equivalence to probability

Let A be an event that depends only on variables in O

To prove $\Pr_{G_1,m_1}[A] = \Pr_{G_2,m_2}[A]$ it suffices to show

- $\models G_1 \simeq^I_O G_2$
- $m_1 =_I m_2$

Proving program equivalence

Goal

$$\models G_1 \simeq^I_O G_2$$

A Relational Hoare Logic

$$\frac{\models c_1 \sim c_2 : \Phi \Rightarrow \Phi' \qquad \models c_1' \sim c_2' : \Phi' \Rightarrow \Phi''}{\models c_1; c_1' \sim c_2; c_2' : \Phi \Rightarrow \Phi''}\text{[R-Seq]}$$

...

Proving program equivalence

Goal

$$\models \mathrm{G}_1 \simeq^I_O \mathrm{G}_2$$

Mechanized program transformations

- Transformation: $T(\mathrm{G}_1, \mathrm{G}_2, I, O) = (\mathrm{G}'_1, \mathrm{G}'_2, I', O')$
- Soundness theorem

$$\frac{T(\mathrm{G}_1, \mathrm{G}_2, I, O) = (\mathrm{G}'_1, \mathrm{G}'_2, I', O') \qquad \models \mathrm{G}'_1 \simeq^{I'}_{O'} \mathrm{G}'_2}{\models \mathrm{G}_1 \simeq^I_O \mathrm{G}_2}$$

- Reflection-based Coq tactic

Proving program equivalence

Goal

$$\vDash G_1 \simeq^{I}_{O} G_2$$

Mechanized program transformations

- Dead code elimination
- Constant folding and propagation
- Procedure call inlining
- Instruction reordering
- Common suffix/prefix elimination

Proving program equivalence

Goal

$$\models G_1 \simeq^I_O G_2$$

A semi-decision procedure for self-equivalence

- Does $\models G \simeq^I_O G$ hold?
- Analyze dependencies to compute I' s.t. $\models G \simeq^{I'}_O G$
- Check that $I' \subseteq I$

Example

Game ElGamal$_0$:

$x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q;$
$(m_0, m_1) \leftarrow \mathcal{A}(g^x);$
$b \xleftarrow{\$} \{0,1\};$
$\zeta \leftarrow g^{xy} \times m_b;$
$b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);$
$d \leftarrow b = b'$

```
inline_r B;
ep;
deadcode;
eqobs_in
```

$\simeq^{\emptyset}_{\{d\}}$

Game DDH$_0$:

$x \xleftarrow{\$} \mathbb{Z}_q;$
$y \xleftarrow{\$} \mathbb{Z}_q;$
$d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$

Procedure $\mathcal{B}(\alpha, \beta, \gamma)$:

$(m_0, m_1) \leftarrow \mathcal{A}(\alpha);$
$b \xleftarrow{\$} \{0,1\};$
$b' \leftarrow \mathcal{A}'(\alpha, \beta, \gamma \times m_b);$
return $b = b'$

The Fundamental Lemma of Game-Playing

Fundamental lemma

If two games G_1 and G_2 behave identically in an initial memory m unless a failure event A fires, then

$$|\Pr_{G_1,m}[A] - \Pr_{G_2,m}[A]| \leq \Pr_{G_{1,2}}[F]$$

The Fundamental Lemma of Game-Playing

Game G_1 :
...
bad ← true; c_1
...

Game G_2 :
...
bad ← true; c_2
...

- $\Pr_{\mathsf{G}_1,m}[A \wedge \neg \mathsf{bad}] = \Pr_{\mathsf{G}_2,m}[A \wedge \neg \mathsf{bad}]$
- $\Pr_{\mathsf{G}_1,m}[\mathsf{bad}] = \Pr_{\mathsf{G}_2,m}[\mathsf{bad}]$

Corollary

$$|\Pr_{\mathsf{G}_1,m}[A] - \Pr_{\mathsf{G}_2,m}[A]| \leq \Pr_{\mathsf{G}_{1,2}}[\mathsf{bad}]$$

Wrapping up

Contributions

- Formal semantics of a probabilistic programming language
- Characterization of probabilistic polynomial-time programs
- A Probabilistic Relational Hoare logic
- Mechanization of common program transformations
- Formalized emblematic proofs: ElGamal, FDH, OAEP

Perspectives

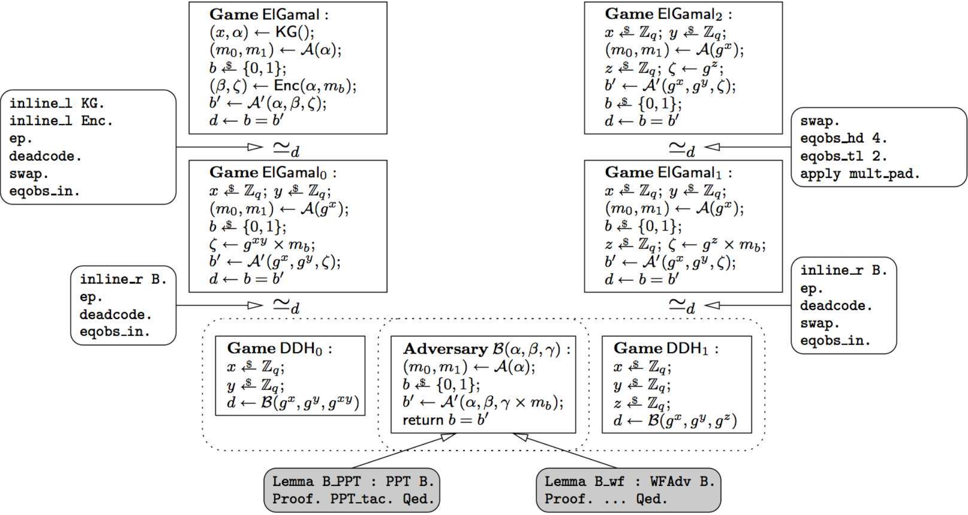
- Overwhelming number of applications: IB, ZK proofs, ...
- Computational soundness of symbolic methods and information flow type systems
- Verification of randomized algorithms

Some statistics

- 6 persons involved
- CertiCrypt: 30,000 lines of Coq, 48 man-months
- Full Domain Hash: 2,500 lines of Coq, 4 man-months (for a person without experience in CertiCrypt)

Questions

ElGamal encryption



$$\left|\Pr_{\mathsf{ElGamal}}[b = b'] - \frac{1}{2}\right| = \left|\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d]\right|$$

Observational equivalence

$$\models G_1 \sim G_2 : \Psi \Rightarrow \Phi \stackrel{\text{def}}{=} m_1\ \Psi\ m_2 \Rightarrow [\![G_1]\!]\ m_1 \sim_\Phi [\![G_2]\!]\ m_2$$

Lifting

$$\text{range } P\ \mu \stackrel{\text{def}}{=} \forall f,\ (\forall a,\ P\ a \Rightarrow f\ a = 0) \Rightarrow \mu\ f = 0$$

$$\mu_1 \sim_\Phi \mu_2 \stackrel{\text{def}}{=} \exists \mu,\ \pi_1(\mu) = \mu_1 \wedge \pi_2(\mu) = \mu_2 \wedge \text{range } \Phi\ \mu$$

Small-step semantics

$$
\begin{array}{rcl}
(\mathsf{nil}, m, [\,]) & \rightsquigarrow & \mathsf{unit}\,(\mathsf{nil}, m, [\,]) \\
(\mathsf{nil}, m, (x, e, c, l) :: F) & \rightsquigarrow & \mathsf{unit}\,(c, (l, m.\mathsf{glob})\{[\![e]\!]\ m/x\}, F) \\
(x \leftarrow p(\vec{e});\ c, m, F) & \rightsquigarrow & \mathsf{unit}\,(E(p).\mathsf{body}, (\emptyset\{[\![\vec{e}]\!]\ m/E(p).\mathsf{params}\}, \\
(\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\,(c_1;\ c, m, F) \\
 & & \qquad \text{if } [\![e]\!]\ m = \mathsf{true} \\
(\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\,(c_2;\ c, m, F) \\
 & & \qquad \text{if } [\![e]\!]\ m = \mathsf{false} \\
(\mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) & \rightsquigarrow & \mathsf{unit}\,(c;\ \mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) \\
 & & \qquad \text{if } [\![e]\!]\ m = \mathsf{true} \\
(\mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) & \rightsquigarrow & \mathsf{unit}\,(c', m, F) \\
 & & \qquad \text{if } [\![e]\!]\ m = \mathsf{false}
\end{array}
$$

$$
\begin{array}{rcl}
(x \leftarrow e;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\,(c, m\{[\![e]\!]\ m/x\}, F) \\
(x \xleftarrow{\$} d;\ c, m, F) & \rightsquigarrow & \mathsf{bind}\,([\![d]\!]\ m)(\lambda v.\ \mathsf{unit}\,(c, m\{v/x\}, F))
\end{array}
$$

Denotation

$$[\![S]\!]_0 \stackrel{\text{def}}{=} \text{unit } S \qquad [\![S]\!]_{n+1} \stackrel{\text{def}}{=} \text{bind } [\![S]\!]_n \; [\![\cdot]\!]^1$$

$$[\![c]\!] \; m : M(\mathcal{M}) \stackrel{\text{def}}{=} \lambda f. \; \text{sup} \; \{[\![(c, m, [\;])]\!]_n \; f|_{\text{final}} \mid n \in \mathbb{N}\}$$